Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Sandi Mlinar

Naprava CNC za vrtanje tiskanih vezij

diplomsko delo

visokošolski strokovni študijski program prve stopnje računalništvo in informatika

Mentor: izr. prof. dr. Patricio Bulić

Ljubljana, 2015

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

Izdelajte napravo CNC za vrtanje tiskanih vezij. Napravo naj krmili mikrokrmilnik SMT32F407 z nameščenim operacijskim sistemom za delo v realnem času FreeRTOS. Napravo krmilite s programom, ki ga napišite v C# programskem jeziku z uporabo .NET ogrodja. Program naj omogoča uporabniku popoln nadzor nad napravo, avtomatsko ali ročno krmiljenje posameznih osi ter uvoz koordinat vrtanja iz programa Sprint-Layout.

Izjava o avtorstvu diplomskega dela

Spodaj podpisani Sandi Mlinar z vpisno številko 63080334, sem avtor diplomskega dela z naslovom:

Naprava CNC za vrtanje tiskanih vezij

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom izr. prof. dr. Patricia Bulića,
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela,
- soglašam z javno objavo elektronske oblike diplomskega dela na svetovnem spletu preko univerzitetnega spletnega arhiva.

V Ljubljani, dne 7. septembra 2015 Podpis avtorja:

Zahvaljujem se mentorju izr. prof. dr. Patriciu Buliću za nasvete in pomoč.

Zahvaljujem se svoji družini, ki me je spodbujala med študija.

Zahvaljujem se svoji punci, ki me je spodbujala in motivirala, da sem dokončal diplomo.

Kazalo

Seznam uporabljenih kratic

kratica	angleško	slovensko
NC	Numerical control	Numerično krmiljenje
CNC	Computer numerical control	Računalniško numerično krmiljenje
CAD	Computer-aided design	Računalniško podprto načrtovanje
CAM	Computer-aided manufacturing	Računalniška podprta proizvodnja
PCB	Printed circuit board	Tiskano vezje
IDE	Integrated development environment	Vgrajeno razvojno okolje
RTOS	Real Time Operating System	Operacijski sistem realnega časa
GPOS	General-Purpose Operating System	Splošno namenski operacijski sistem
FPU	Floating-point unit	Enota s plavajočo vejico
RISC	Reduced Instruction Set Computer	Računalnik z okrnjenim naborom inštrukcij
RAM	Random Access Memory	Bralno-pisalni pomnilnik
USART	Universal Synchronus Asyncronous Receiver Transmitter	Univerzalni sinhroni asinhroni sprejemnik oddajnik
PWM	Pulse Width Modulation	Pulzno širinska modulacija

Povzetek

V diplomskem delu je predstavljen postopek za izdelavo računalniško krmiljene naprave CNC za vrtanje tiskanih vezij. Predstavljena je realizacija strojnega dela naprave, uporabljeni materiali in postopki izdelave. Za premike obdelovalnega orodja uporabljamo koračne motorje, ki nam omogočajo želeno preciznost premika. Zaradi mehanskih premikov stroja so na samem ogrodju naprave varovalna končna stikala, ki ustavijo delovanje naprave v primeru napak.

Napravo krmili mikrokrmilnik STM32F407 z nameščenim operacijskim sistemom za delo v realnem času FreeRTOS. Napravo krmilimo s programom, ki smo ga napisali v C# programskem jeziku z uporabo .NET ogrodja. Program omogoča uporabniku popoln nadzor nad napravo, avtomatsko ali ročno krmiljenje posameznih osi ter uvoz koordinat vrtanja iz programa Sprint-Layout. Program s krmilnikom komunicira preko vmesnika USART *(angl. Universal Synchronous Asynchronous Receiver Transmitter)*, za sporazumevanje pa uporabljata predefinirane ukaze.

Ključne besede: CNC, mikrokrmilnik, naprava, tiskano vezje, C, C#, FreeRTOS.

Abstract

This thesis presents a method for producing a computer controlled CNC device for drilling printed circuit boards. The presented work is the realization of machine equipment, materials and manufacturing processes. Stepper motors are used for moving machining tool, which allow us to achieve the desired precision. Due to the mechanical movements of the machine, end switches are installed which in case of error stops the device.
The device is controlled by STM32F407 microcontroller running FreeRTOS operating system. The device is controlled by a program that was written in the C# programming language using the. NET Framework. The program allows the user complete control over the device, automatic or manual control of individual axes and the ability to import drilling coordinates from Sprint-Layout program. The program communicates with the controller over USART (Universal Synchronous Asynchronous Receiver Transmitter) interface and the predefined commands are used for communication.

Keywords: CNC, microcontroller, device, circuit board, C, C#, FreeRTOS.

Poglavje 1

Uvod

V sodobnem času človek uporablja vedno več naprav. Vsem napravam je skupno to, da za delovanje potrebujejo tiskano vezje, na katerem se nahajajo vsi potrebni elementi za delovanje naprave. Pertinaks je osnova za izdelavo tiskanega vezja. To je plošča, ki mehansko podpira in električno poveže elektronske komponente s prevodnimi potmi, katere so jedkane iz bakrene površine na neprevodni plošči. Pertinaks plošča je lahko enoplastna (ena bakrena plast), dvoplastna (dve bakreni plasti) ali večplastna (zunanje in notranje bakrene plasti). Obstaja več vrst in načinov izdelave tiskanih vezij, ki so v sodobni industriji večinoma avtomatizirani in navadnemu človeku nedostopni. Vendar pa se domače, neprofesionalne izdelave tiskanega vezja lahko lotimo kar sami. Potrebujemo natisnjeno predlogo želenega tiskanega vezja, ki ga z laserskim tiskalnikom natisnemo na povoskan papir. To predlogo z laminatorjem prenesemo na pertinaks, nato sledi postopek jedkanja. Naslednji korak je vrtanje lukenj, ki sem se odločil avtomatizirati.

Pri domači izdelavi tiskanega vezja ponavadi vrtamo ročno. Ta korak je lahko precej naporen, če smo načrtovali tiskano vezje večje velikosti z veliko elektronskimi elementi. Ko načrtujemo tiskano vezje s programom, nam ta program ponavadi omogoča izvoz datoteke s koordinatami lukenj. Če imamo računalniško krmiljeno napravo CNC (angl. Computer Numerical Control) za vrtanje, lahko to datoteko uporabimo za avtomatizacijo postopka.

Idejo za diplomsko delo smo dobili že v srednji šoli, kjer smo pri praktičnem izobraževanju izdelali kar nekaj tiskanih vezij. Tako smo se odločili narediti napravo CNC za vrtanje tiskanih vezij. Napravo smo izdelali od ideje do končnega delujočega prototipa. Pri strojnem delu naprave smo uporabili nekaj elementov iz starih računalnikov in tiskalnikov. Krmilni del smo realizirali z uporabo STM32F4Discovery (Poglavje 4.4) razvojne plošče. Razvojna plošča je primerna zaradi velikega števila vhodno-izhodnih nožic in dobre podpore perifernih naprav. Ima tudi možnost uporabe operacijskega sistema za delo v realnem času. Izdelali smo tudi tiskano vezje, ki povezuje vse periferne enote, kot so končna stikala, koračni motorji ... Razvili smo tudi krmilni program, ki uporabniku omogoča celovito upravljanje z napravo. Za razvoj programa smo uporabili C# ter ogrodje .NET.

Drugo poglavje opisuje zgodovino naprav CNC, vrste ter načine krmiljenja naprav CNC. Tretje poglavje opisuje funkcionalnosti izdelane naprave ter njeno uporabo. Opisuje tudi programska orodja, ki smo jih uporabili za izvedbo programskega dela. Četrto poglavje opisuje strojni del naprave, uporabljene mehanske dele in opis krmilne plošče. Peto poglavje opisuje programski del naprave, tako na mikrokrmilniku, kot tudi sam program za krmiljenje.

Poglavje 2

Naprava CNC

Naprava CNC - računalniško numerično krmiljenje [2] deluje na kartezičnem koordinatnem sistemu (x, y, z) za 3D nadzor gibanja. Deli projekta so lahko izdelani z računalniškim programom CAD/CAM [3] in nato avtomatsko obdelani do končnega produkta.

Naprava je krmiljena z naprednim računalniškim sistemom, ki s pomočjo zapletenih izračunov ukaze sproti preverja ter izvaja natančne in zahtevne gibe obdelovanca ter orodij. S sprotno kontrolo ukazov nam zagotavlja zaščito naprave, prav tako pa preverja pravilno izvajanje premikov krmilne naprave. S tem nam zagotovi natančnost izvajanja operacij oziroma prekinitev izvajanja zaradi varnostnih razlogov.

Naprave CNC obstajajo v različnih izvedbah, od tako imenovanih namiznih naprav CNC do velikih industrijskih naprav CNC. Kljub številnim konfiguracijam pa imajo nekaj specifičnih delov:

- namenski CNC krmilnik,
- vreteno motorja,
- servo motorje,
- koračne motorje,
- servo ojačevalniki,

- linearna vodila.

2.1 Zgodovina naprav CNC

Zgodovina numeričnega krmiljenja se je začela z avtomatizacijo navadnih mehanskih strojev. Napredek in razvoj na področju elektrotehnike in elektronike sta omogočila razvoj visoko avtomatiziranih numerično krmiljenih NC (angl. Numerical Control) [4] strojev. Prve stroje so začeli izdelovati v letu 1940. Temeljili so na obstoječih orodjih, ki so bili nadgrajeni oziroma krmiljeni z motornim pogonom, kar je avtomatiziralo obdelavo. Stroje krmilimo z nizom ukazov, ki jih neposredno ali pa prek nosilnih medijev vnesemo v napravo. Naprava ukaze prebere, obdela in izvede, vendar pa dinamično spreminjanje ukazov ni več mogoče. Velik preskok je bila uvedba strojev CNC, ki so računalniško vodeni.

2.2 Vrste naprav CNC

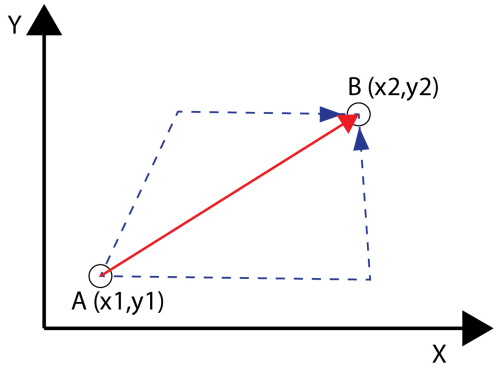
V osnovi delovanja in krmiljenja naprav CNC ni velikih razlik, le da so določeni deli prilagojeni numeričnemu krmiljenju ter kontroliranju procesa krmiljenja. Glede na vrsto obdelave jih delimo na:

- numerično krmiljene stroje za odrezovanje,
- numerično krmiljene stroje za izsekovanje in prebijanje,
- numerično krmiljene stroje za preoblikovanje,
- numerično krmiljene stroje za ostale postopke razdvajanja,
- numerično krmiljene stroje, ki ne spadajo v skupino NC obdelovalnih strojev.

2.3 Načini krmiljenja

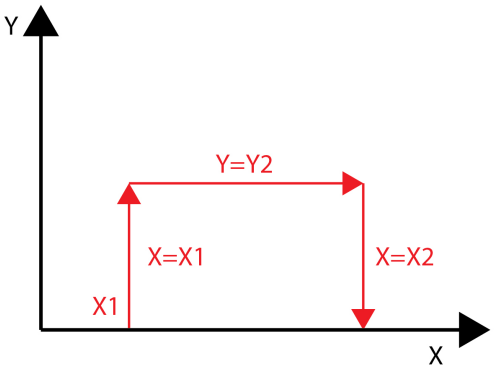
Obstaja več načinov krmiljenja. Med tri najpogostejše spadajo:

- krmiljenje od točke do točke (uporablja se predvsem za vrtalne naprave CNC, kjer je obdelovanje točkovno, kar pomeni, da obdelovano orodje med premikanjem ne obdeluje) (Slika 2.1),



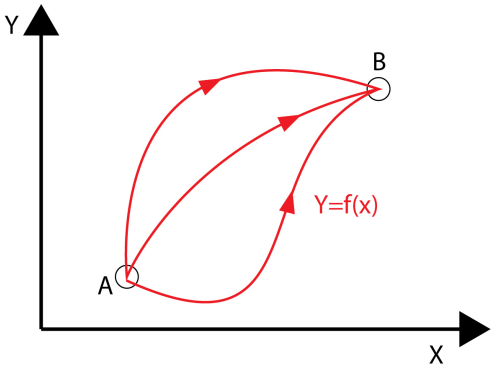
Slika 2.1: Točkovno krmiljenje [5].

- krmiljenje po ravnih linijah (obdelovalno orodje med samim premikanjem od točke a do točke b že obdeluje, običajno pa se orodje premika vzporedno z osmi) (Slika 2.2),



Slika 2.2: Linijsko krmiljenje [6].

- konturno krmiljenje (obdelovalno orodje se lahko premika po dveh ali treh oseh, gibanje med orodjem in obdelovancem pa se lahko neprestano menja) (Slika 2.3).



Slika 2.3: Konturno krmiljenje [7].

Poglavje 3

Funkcionalnosti in uporabljene tehnologije

Ideja za izdelavo računalniško krmiljene naprave CNC za vrtanje tiskanih vezij se nam je porodila že v času srednje šole, kjer smo morali ročno vrtati tiskana vezja. Da bi ta proces vrtanja vezij nadgradili in avtomatizirali, smo se odločili narediti napravo CNC za vrtanje le-teh.

3.1 Funkcionalnosti naprave

Glede na ideje, ki smo jih imeli, ima naprava sledeče funkcionalnosti:

- ročno pozicioniranje na želeno mesto,
- avtomatsko pozicioniranje na privzeto mesto,
- ročno premikanje obdelovalnega orodja,
- zaznavanje napačnih pozicij s končnimi stikali,
- avtomatsko delovanje.

3.2 Programski jezik C

Programski jezik C [8] je nizkonivojski standardizirani računalniški programski jezik tretje generacije za splošno rabo. Razvil ga je Dennis Ritchie med letoma 1969 in 1973 sprva za uporabo na računalnikih PDP-11 [9] z operacijskim sistemom UNIX. Razvit je bil za delo pod operacijskim sistemom Unix, kmalu pa se je pojavilo veliko zanimanje za verzijo, ki bi jo lahko poganjali pod MS-DOS. V poznih sedemdesetih letih so izvedli različice jezika C za širok razpon računalnikov, miniračunalnikov in mikroračunalnikov, vključno z osebnim računalnikom IBM PC.

3.3 Programski jezik C#

Programski jezik C#, ki ga je razvilo podjetje Microsoft pod vodstvom Andersa Hejlsberga leta 2001, je razmeroma nov programski jezik. Podpira vse funkcionalnosti ogrodja .NET Framework. Je preprost, moderen, objektno usmerjen jezik in programerja sili k temu, da uporablja objektno in komponentno orientirane programerske discipline. Zgrajen je na sintaksi in semantiki programskega jezika C++. Zadnja stabilna verzija je 5.0, ki je bila objavljena 15. avgusta 2012 [10].

3.4 Operacijski sistem FreeRTOS

Operacijski sistem FreeRTOS [11] je operacijski sistem za delo v realnem času, prilagojen za vgrajene sisteme z malo pomnilnika. Od splošno namenskih operacijskih sistemov GPOS [12] (angl. General-Purpose Operating System) se razlikuje predvsem po manjši porabi pomnilnika, hitrejšem in zanesljivejšem izvajanju ter po veliki stopnji prilagodljivosti. Osnovne naloge sistema RTOS so obdelava prekinitev, krmiljenje izvajanja procesov, medprocesna komunikacija, sinhronizacija in krmiljenje perifernih naprav. Omogoča razdelitev aplikacije v manjše programske enote, imenovane opravila. Večina izvorne kode operacijskega sistema je napisana v programskem

jeziku C. Zbirni jezik je uporabljen le pri npr. menjavi kontekstne vsebine.

3.5 Uporabljena orodja

Za razvoj programa za krmiljenje in programa na mikrokrmilniku ter tiskanega vezja smo uporabili naslednja orodja:

- Sprint-Layout 5.0,
- CooCox CoIDE,
- Microsoft Visual Studio 2010.

3.5.1 Sprint-Layout 5.0

Sprint-Layout [13] je orodje za oblikovanje tiskanih vezij PCB [16] (angl. Printed Circuit Board) tiskanih vezij. Ponuja vse potrebne funkcije in knjižnjico z najbolj pogostimi komponentami za izdelavo po meri izdelanega tiskanega vezja. Omogoča izvoz v več različnih formatov, ki jih lahko uporabimo za fizično izdelavo tiskanega vezja. Eden izmed njih je Excellon [17]. To je format, ki ga lahko uporabimo za krmiljenje vrtalne naprave CNC. Vsebuje velik nabor ukazov [18], katere znajo sprocesirati nekatere naprave CNC. Prav ta format sem uporabil pri svoji napravi CNC.

3.5.2 CooCox CoIDE

Razvojno okolje CooCox CoIDE [14] je odprtokodno okolje za razvoj aplikacij za različne družine mikrokrmilnikov. CooCox je bil zagnan kot projekt Embest Info.Tech.Co skupaj z univerzo Wuhan v začetku leta 2009. Z leti je rastel in postal svetovna neodvisna znamka razvojnih orodij.

3.5.3 Microsoft Visual Studio 2010

Microsoft Visual Studio [15] je vgrajeno razvojno okolje, ki ga je razvilo podjetje Microsoft leta 2010. Večinoma se uporablja za razvoj aplikacij za

operacijski sistem Microsoft Windows, prav tako pa se uporablja za razvoj spletnih aplikacij in storitev. Visual Studio uporablja Microsoftove razvojne platforme, kot so Windows API, Windows Forms ipd. Podpira različne programske jezike, kot so C++, Visual C++, Visual Basic, C#.

Poglavje 4

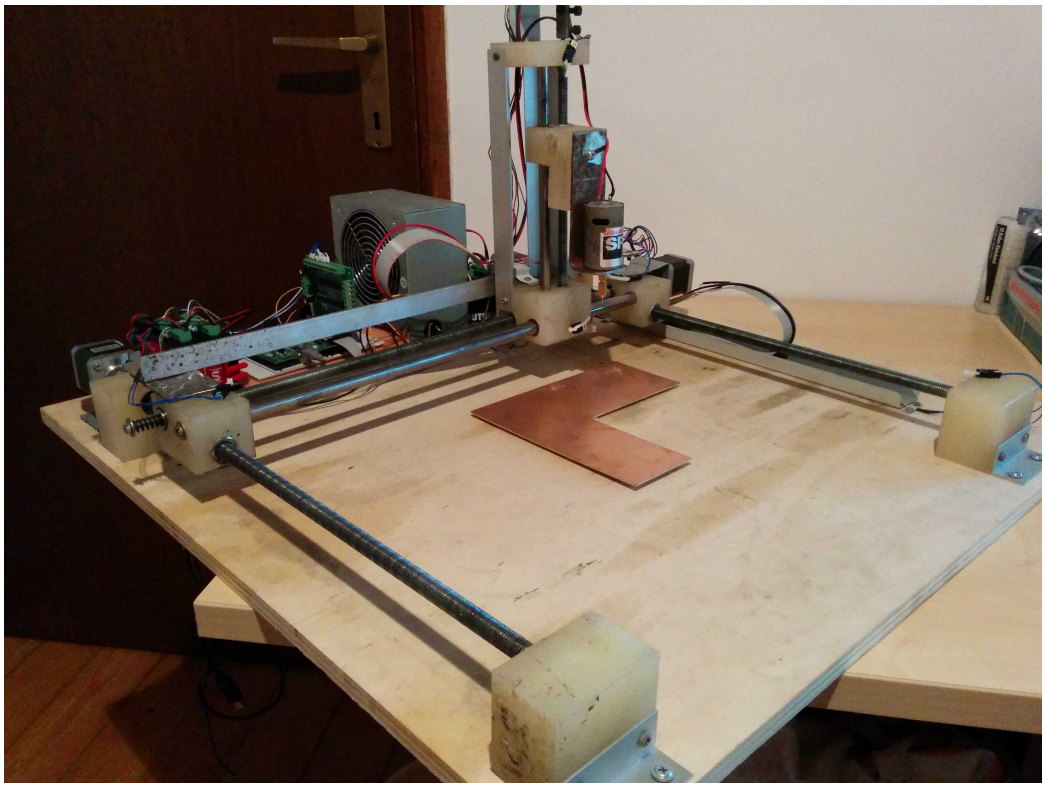
Strojni del

4.1 Ogrodje naprave

Za osnovno mehansko delovanje naprave smo potrebovali ogrodje, ki je kompaktno ter predvsem dovolj veliko za obdelovanje tiskanih vezij različnih velikosti. Za osnovni element ogrodja smo uporabili vezano ploščo debeline 1,5 cm, ki je dovolj trdna, da se ne krivi. Na vezano ploščo je pritrjeno ogrodje, ki omogoča premikanje obdelovalnega orodja v smeri X, Y in Z. Premikanje po vseh treh oseh je omogočeno z navojnimi palicami. Na Y in Z je dodatna vodilna palica. Palice so vpete v ogrodje iz polietilena prek ležajev, s katerimi smo zmanjšali trenje. Na Sliki 4.1 je prikazana naprava CNC.

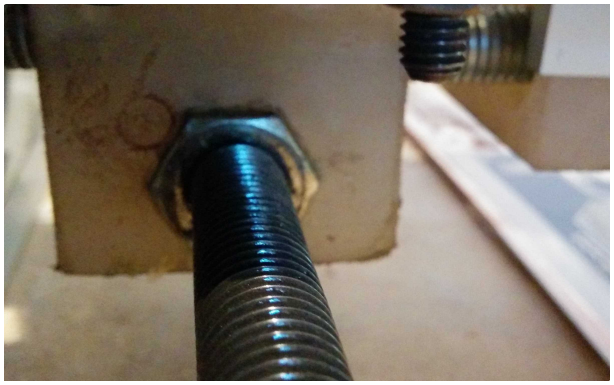
4.2 Mehanski deli

Napravo sestavljajo različni mehanski deli. Za premikanje po oseh smo uporabili navojne palice M12 premera 12 mm (Slika 4.2). Te palice so preko matic vpete v ogrodje iz polietilena (Slika 4.3), kar nam omogoča premikanje obdelovalnega orodja. Polietilen smo izbrali zato, ker ga je razmeroma lahko obdelovati in oblikovati. Na koncih palic so zaradi zmanjšanja trenja palice vpete v polietilen prek ležajev. Na enem koncu je v sredino palice izvrtana luknja, kamor je vpet koračni motor (Slika 4.4). Na Y in Z osi imamo



Slika 4.1: Naprava CNC.

Slika 4.2: Navojna palica.

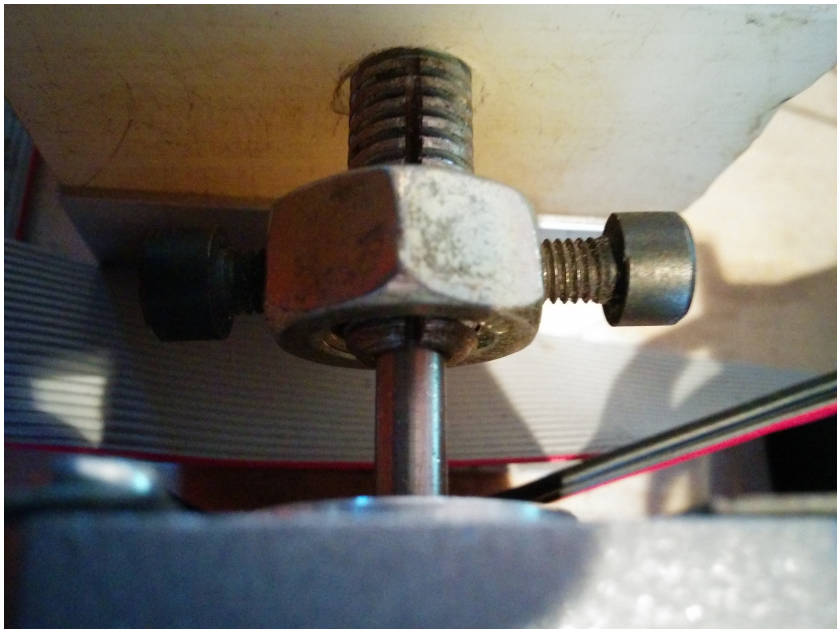


Slika 4.3: Vpetje navojnih palic na ogrodje.

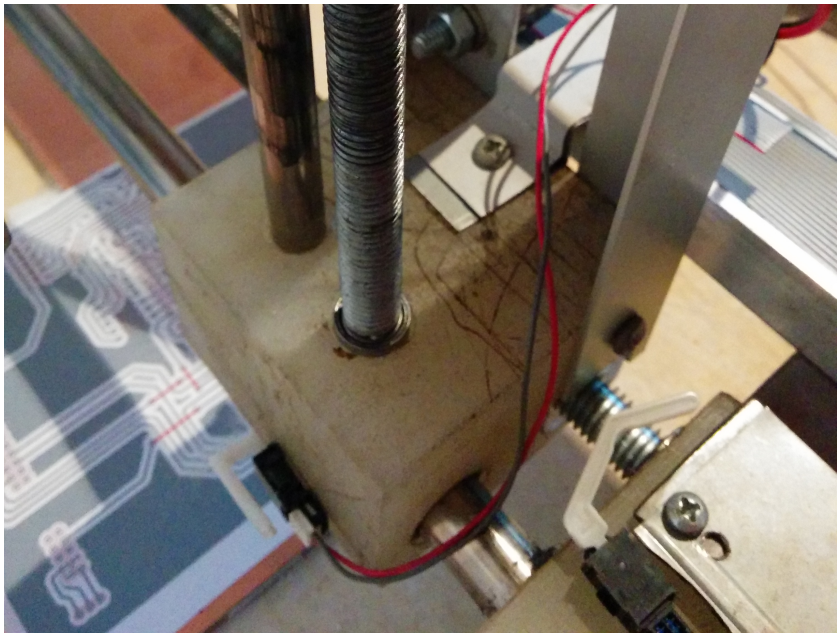
zaradi večje stabilnosti orodja dodatno vodilno palico. Vodilna palica nam omogoča premikanje in stabilizacijo obdelovalnega orodja z le eno navojno palico (Slika 4.5). Obdelovalno orodje je motor, ki deluje na 12 V. Na os ima vpeto vpenjalno glavo, v katero lahko vpnemo sveder velikosti od 0,7 mm do 3,2 mm (Slika 4.6).

4.2.1 Koračni motorji in krmilniki

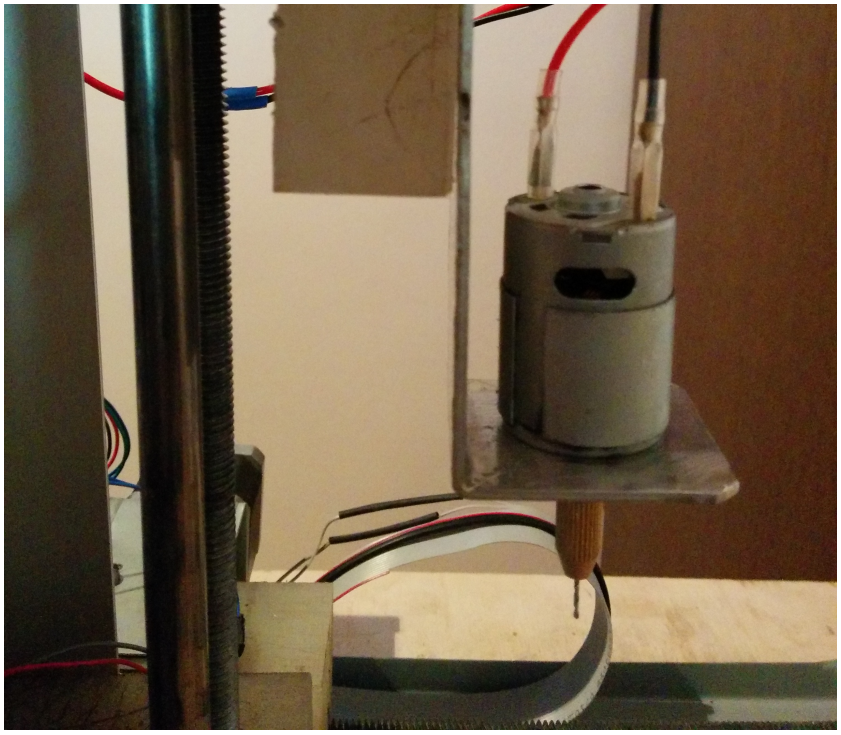
Koračni motorji so elektromehanske naprave, ki omogočajo pretvarjanje digitalnega podatka v mehanski premik. Pri vzbujanju po programiranem zaporedju se rotor v diskretnih korakih ustrezno zavrti v želeno smer. Koračni motorji se večinoma uporabljajo kot izvršilni člen pri krmiljenju z digitalnimi vezji. Njihova prednost je v preciznem in hitrem pozicioniranju.



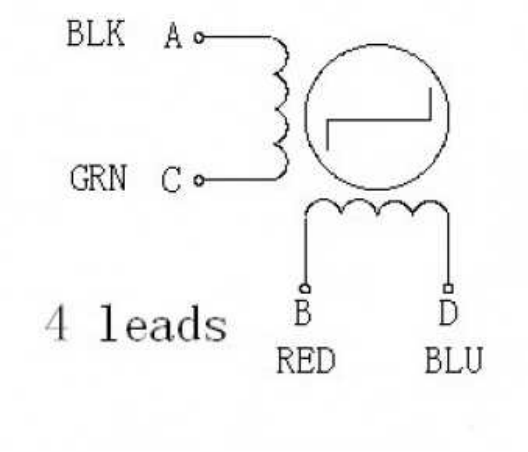
Slika 4.4: Vpetje motorja na navojne palice.



Slika 4.5: Vodilne palice na Y in Z-osi.



Slika 4.6: Obdelovalno orodje.



Slika 4.7: Zgradba koračnega motorja in priklop signalnih linij [20].

Obstaja več vrst koračnih motorjev. Glede na potrebe naprave lahko koračni motor krmilimo v načinu:

- polnega koraka (angl. full stepping),
- polovičnega koraka (angl. half stepping)
- in mikrokoraka (angl. microstepping).

Za potrebe izdelave naprave CNC smo uporabili unipolarne dvofazne koračne motorje NEMA17 [19]. Sestavljen je iz dveh navitij na en par polov. Vsako navitje lahko vzbujamo neodvisno od drugih navitij. Najenostavnejši način krmiljenja unipolarnega koračnega motorja s polnim korakom je prikazan v Tabeli 4.1, način s polovičnim korakom pa v Tabeli 4.2, pri čemer so A, B, C in D signali na vhodu koračnega motorja na Sliki 4.7.

Pri naši napravi smo uporabili mikrokora. To je način krmiljenja motorja, kjer motor teče najbolj gladko in tiho. Pri tem je tok v tuljavah izmeničen oziroma sinusen. Pri napravah s preciznim pozicioniranjem je mikrokorak zelo pomembna funkcionalnost, saj pripomore k natančnejšem premikanju in pozicioniranju.

Za gonilnik koračnih motorjev smo uporabil dvofazni hibridni gonilnik DQ420MA [21] (Slika 4.8). Podpira možnost napajanja od 12 36 V, ima

	Signali			
Korak	A	C	B	D
1	1	0	1	0
2	1	0	0	1
3	0	1	0	1
4	0	1	1	0

Tabela 4.1: Krmiljenje koračnega motorja s polnim korakom.

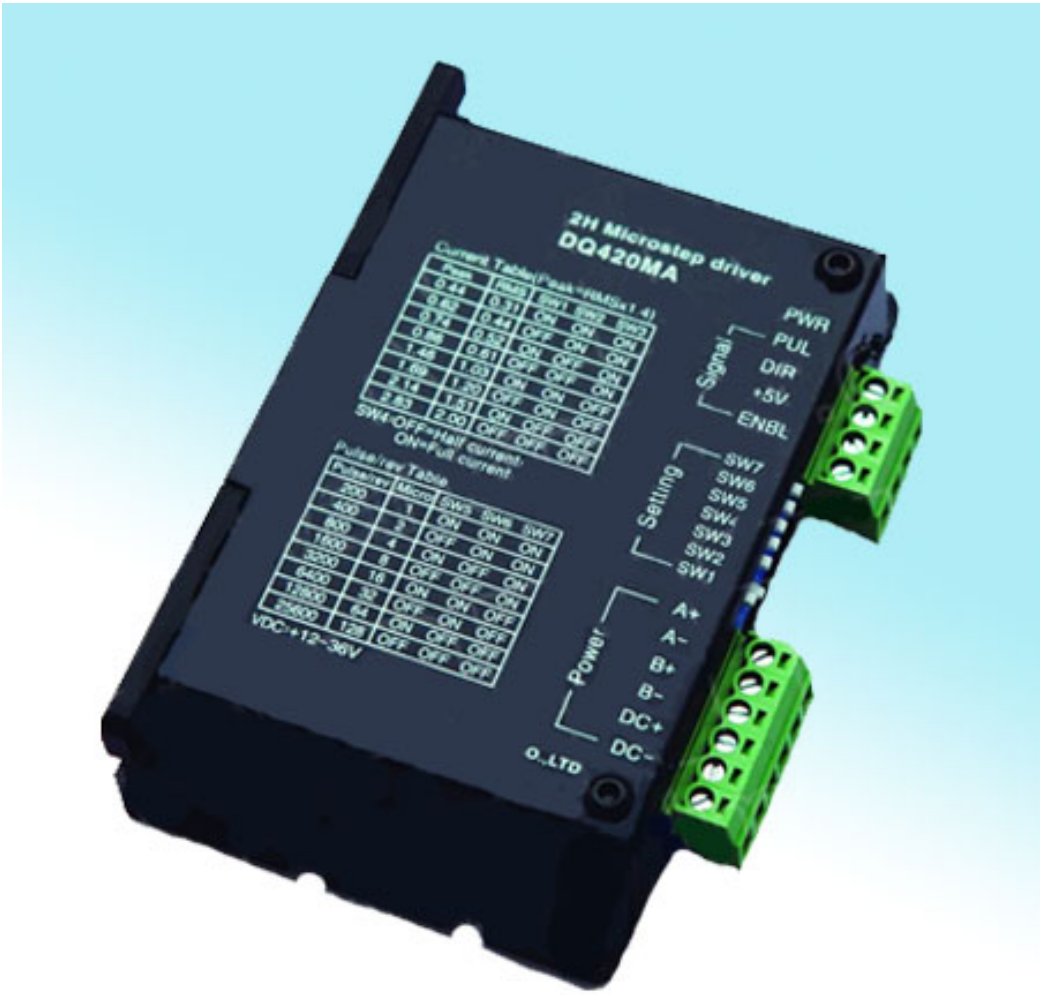
	Signali			
Korak	A	C	B	D
1	1	0	1	0
2	1	0	0	1
3	1	0	0	1
4	0	0	0	1
5	0	1	0	0
6	0	1	0	0
7	0	1	1	0
8	0	0	1	0

Tabela 4.2: Krmiljenje koračnega motorja s polovičnim korakom.

		Stikala		
Delitelj	`Korakov/obrat`	`SW5`	`SW6`	`SW7`
`1`	200	ON	ON	ON
`2`	400	OFF	ON	ON
`4`	800	ON	OFF	ON
`8`	1600	OFF	OFF	ON
`16`	3200	ON	ON	OFF
`32`	6400	OFF	ON	OFF
`64`	12800	ON	OFF	OFF
`128`	25600	OFF	OFF	OFF

Tabela 4.3: Tabela izbire resolucije korakanja.

optično ločene signalne linije ter omogoča izbiro resolucije korakanja (prikazano v Tabeli 4.3) s stikali SW5, SW6, SW7 (Slika 4.8).



Slika 4.8: Gonilnik koračnega motorja.

4.3 Krmilno vezje

Krmilno vezje povezuje vso zunanjo periferijo z mikrokrmilnikom. Povezuje:

- končna stikala,
- napajalni del,
- priklop koračnih motorjev,
- priklop RS232 komunikacije,
- priklop obdelovalnega orodja
- in priklop mikrokrmilnika.

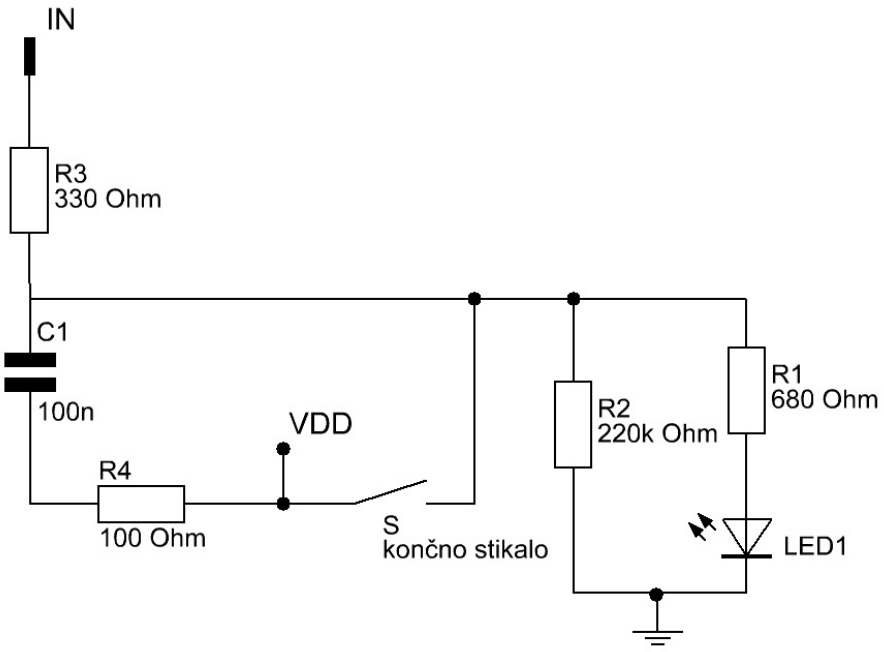
4.3.1 Končna stikala

Pri napravi smo uporabili končna stikala na vseh oseh premika. So ključnega pomena za zaustavitev premika orodja, če pride do napake pri izračunu koordinat, uporabljena pa so tudi pri določanju začetne pozicije (pozicije 0:0:0) obdelovalnega orodja. Končna stikala so bila vzeta iz starega LaserJet 6L tiskalnika (Slika 4.9). Na Sliki 4.10 je prikazana električna shema priklopa končnih stikal.

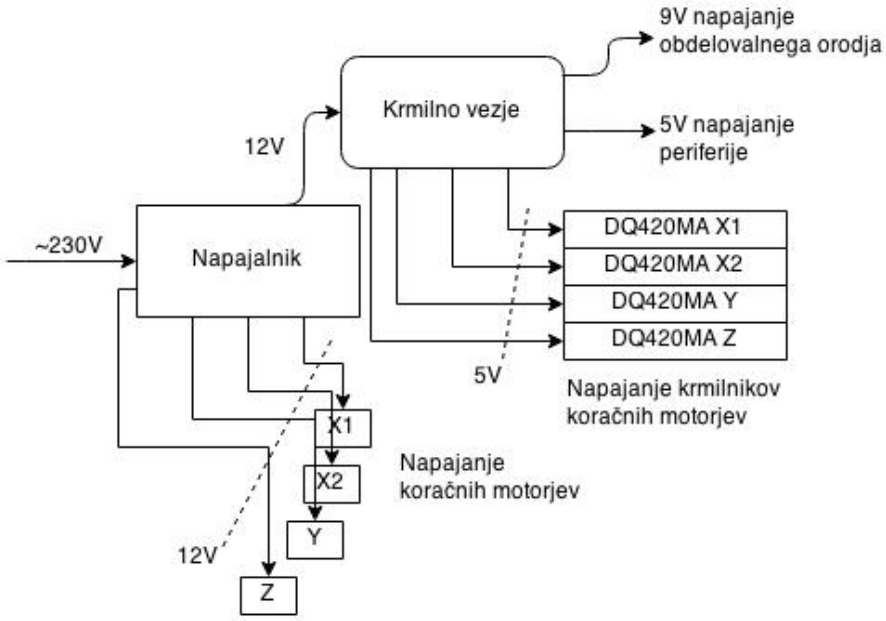
4.3.2 Napajalni del

Za napajanje naprave smo uporabili napajalnik iz starega osebnega računalnika. Predelali smo ga tako, da imamo na izhodu le tri pare žic (tri žice GND in tri žice 12 V). Na Sliki 4.11 je prikazana napajalna shema naprave. Napajalnik se napaja iz omrežja z 230 V izmenične napetosti. To napetost se potem v napajalniku pretvori v enosmerno. Z 12 V iz napajalnika neposredno napajamo koračne motorje in krmilno vezje, kjer se generira 5 V za napajanje periferije in za napajanje krmilnikov koračnih motorjev. 5 V se generira z napetostnim regulatorjem LM7805 [22], ki vhodno napetost zniža na konstantnih 5 V. Na

Slika 4.9: Končno stikalo.



Slika 4.10: Električna shema priklopa končnih stikal.

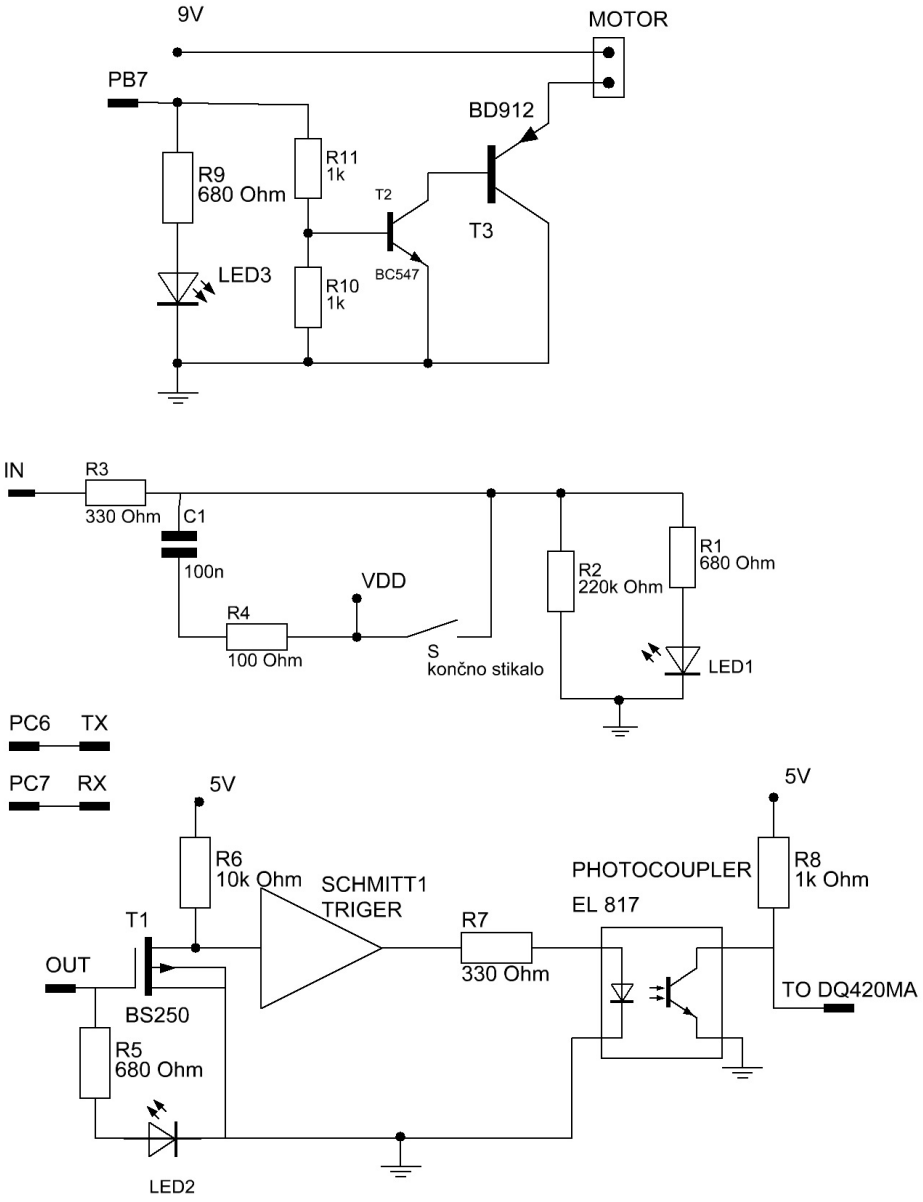


Slika 4.11: Shema napajalnega dela.

krmilnem vezju prav tako generiramo 9 V za napajanje obdelovalnega orodja. Električna shema krmilnega vezja je prikazana na Sliki 4.12.

4.3.3 Priklop koračnih motorjev

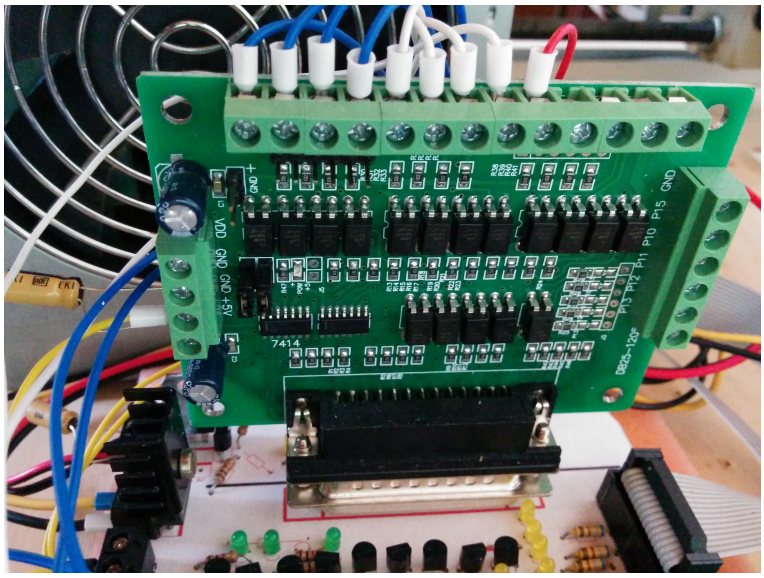
Koračni motorji oz. krmilniki koračnih motorjev so na krmilno vezje priklopljeni prek dodatnega vezja (Slika 4.13), ki smo ga dobili v paketu z motorji. Vezje ima integrirane optične izolatorje C-klase [23], ki nam omogočajo električno izolacijo od krmilnega vezja. Naloga vezja je tudi zaščita signalnih linij v primeru povratnega udara napetosti na signalnih linijah. Na to vezje z ene strani priklopimo signalne linije iz krmilnikov koračnih motorjev, z druge strani pa pripeljemo naše signalne linije. Električna shema povezave signalov od mikrokrmilnika do krmilnika koračnega motorja pa je prikazana na Sliki 4.14.



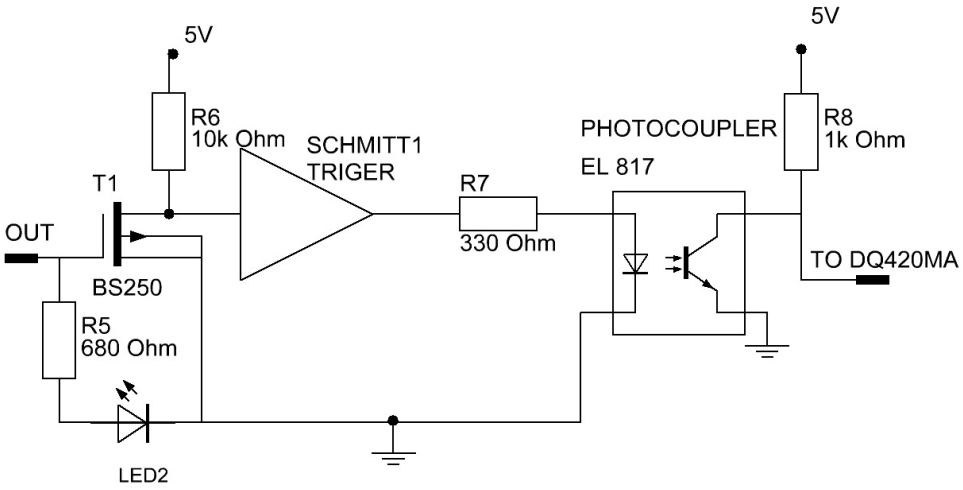
Stikalo (IN)	Vhodni pin (STM32F4)
X1	PA0
X2	PA1
Y1	PA2
Y2	PA3
Z1	PA4
Z2	PA5

Signal (OUT)	Izhodni pin (STM32F4)
XDIR	PB0
ZDIR	PB1
YDIR	PB3
XDIR	PB6
XPWM	PA6
YPWM	PD15
ZPWM	PA8

Slika 4.12: Električna shema krmilnega vezja.



Slika 4.13: Dodatno vezje.



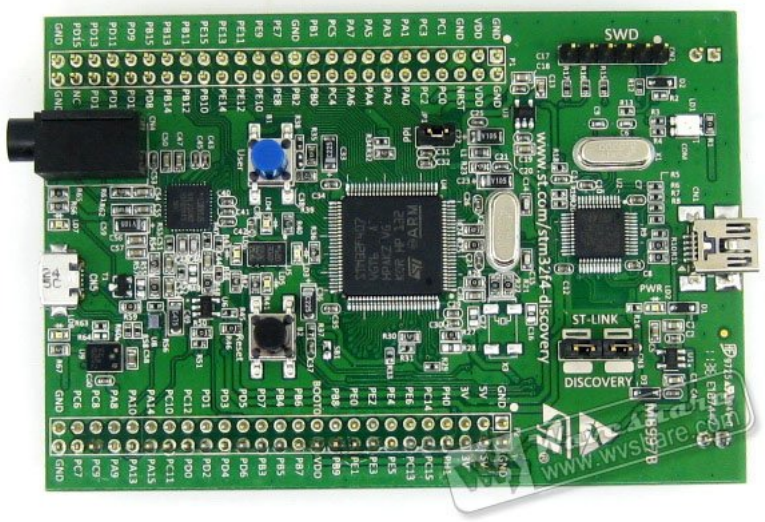
Slika 4.14: Električna shema ene signalne linije.

4.4 Opis mikrokrmilnika

STM32 [24] je družina 32-bitnih mikrokrmilnikov, ki so razdeljeni v več serij. Tem serijam je skupno 32-bitno ARM jedro, imajo pa različno količino statičnega RAM-a, flash pomnilnika in podprte periferije.

Za našo napravo smo uporabili mikrokrmilnik STM32F407. Vsebuje Cortex-M4 jedro in bazira na ARMv7-M [25] arhitekturi. Prednosti teh jeder so integrirani prekinitveni krmilniki NVIC (angl. Nested Vector Interrupt Controller), ki so oblikovani za nizkolatentne čase, učinkovitost in nastavljivost. K temu pripada še krmilnik WIC (angl. Wakeup Interrupt Controller), ki skrbi za prehajanje v stanje spanja, v katerem mikrokrmilnik porabi izjemno malo energije. Ta arhitektura podpira ukaze Thumb2, kjer se kombinirajo 16 in 32-bitni ukazi [25]. Podpira tudi preračunavanje števil s plavajočo vejico.

Razvojna plošča STM32F4Discovery [1] (Slika 4.15) ima 1 MB Flash spomina, 192 KB RAM spomina, napaja se lahko prek USB-ja ali prek zunanje linije. Plošča ima vgrajen tudi senzor pospeška in gibanja, zagotavlja pa tudi podporo za digitalno procesiranje signalov. Za razhroščevanje in lažje programiranje ima razvojna plošča vgrajen programator ST-LINK/V2 [26], ki ga z računalnikom povežemo prek vmesnika USB.



Slika 4.15: Razvojna plošča STM32F4Discovery.

Poglavje 5

Programski del

Programski del diplomskega dela je obsegal programiranje mikrokrmilnika ter programiranje aplikacije, s katero sta omogočena komunikacija ter krmiljenje naprave. Na mikrokrmilniku smo vzpostavili FreeRTOS ter vse pripadajoče prekinitveno-servisne rutine, ki so potrebne za komunikacijo ter delovanje programa. Pri programiranju aplikacije smo morali implementirati uvoz določenega tipa datoteke, v kateri so shranjene koordinate za vrtanje tiskanega vezja. Prav tako smo implementirali komunikacijo s programom na mikrokrmilniku ter vse pripadajoče akcije za upravljanje in nadzor naprave.

5.1 Program na mikrokrmilniku

Programski del na mikrokrmilniku vsebuje dve glavni opravili. Prvo je zadolženo za pravilno in usklajeno vklapljanje prekinitveno servisnih rutin. Te rutine skrbijo za pulzno-širinsko modulacijo (PWM [27]) in krmiljenje koračnih motorjev. Nadzira tudi stanje končnih stikal ter upravlja s smerjo premika posamezne osi. Glavna naloga drugega opravila je nadzor nad komunikacijo s krmilnim programom ter sporočanje koordinat premika prvemu opravilu. Opravili med seboj komunicirata z uporabo sporočilne vrste.

Z mikrokrmilnikom prek izhodnih pinov upravljamo več različnih komponent. Te so:

- vhodni signali končnih stikal,
- izhodni signali za krmiljenje smeri premika na vseh treh oseh,
- izhodni signal za vklop/izklop obdelovalnega orodja.

Ker imamo v sistemu šest vhodnih pinov za krmiljenje končnih stikal ter štiri pine za krmiljenje smeri premikanja na treh oseh, smo za inicializacijo pinov napisali funkcije, ki nam olajšajo inicializacijo.

Pine smo združili v dva enumeratorja (Koda 5.1). Enumerator `OutSignal_TypeDef_t` definira pine, s katerimi krmilimo smer premika posamezne osi, enumerator `Switches_TypeDef_t` pa definira vhodne pine, na katere so priklopljena končna stikala.

```
typedef enum OutSignal_TypeDef_e
{
  X_DIRECTION1 = 0,
  Z_DIRECTION  = 1,
  Y_DIRECTION  = 3,
  X_DIRECTION2 = 6
} OutSignal_TypeDef_t;

typedef enum Switches_TypeDef_e
{
  SWITCH_X_1 = 0,
  SWITCH_X_2 = 1,
  SWITCH_Y_1 = 2,
  SWITCH_Y_2 = 3,
  SWITCH_Z_1 = 4,
  SWITCH_Z_2 = 5,
} Switches_TypeDef_t;
```

Koda 5.1: Enumeratorja vhodno/izhodnih pinov.

5.1.1 Inicializacija izhodnih pinov

V kodi 5.2 je prikazana nadaljnja inicializacija enega izhodnega pina. V spremenljivki `OUTn` imamo definirano število vseh izhodnih pinov, v `OUT1_PIN` je shranjena številka prvega pina, `OUT1_GPIO_PORT` vsebuje vrata in `OUT_GPIO_CLK` vsebuje uro vodila. Vse te spremenljivke smo nato shranili v tabelo pinov, vrat in ur (Koda 5.3).

```
#define OUTn  7     /*!< Number of output pins */
#define OUT1_PIN  GPIO_Pin_0                /*!< Pin 0 */
#define OUT1_GPIO_PORT  GPIOB               /*!< Port B */
#define OUT1_GPIO_CLK   RCC_AHB1Periph_GPIOB /*!< Clock */
```

Koda 5.2: Definiranje enega izhodnega pina.

```
/** Table of pins */
const uint16_t OUT_PIN[OUTn] =
{OUT1_PIN, OUT2_PIN, OUT3_PIN, OUT4_PIN,
OUT5_PIN, OUT6_PIN, OUT7_PIN};
/** Table of ports */
GPIO_TypeDef* OUT_PORT[OUTn] =
{OUT1_GPIO_PORT, OUT2_GPIO_PORT, OUT3_GPIO_PORT,
OUT4_GPIO_PORT, OUT5_GPIO_PORT, OUT6_GPIO_PORT,
OUT7_GPIO_PORT};
/** Table of clock ports */
const uint32_t OUT_CLK[OUTn] =
{OUT1_GPIO_CLK, OUT2_GPIO_CLK, OUT3_GPIO_CLK,
OUT4_GPIO_CLK, OUT5_GPIO_CLK, OUT6_GPIO_CLK,
OUT7_GPIO_CLK};
```

Koda 5.3: Tabela definicij pinov.

Za olajšanje inicializacije večjega števila izhodnih pinov smo napisali funkcijo `STM_EVAL_OutInit(OutSignal_TypeDef_t Out_signal)`. Funkcija kot

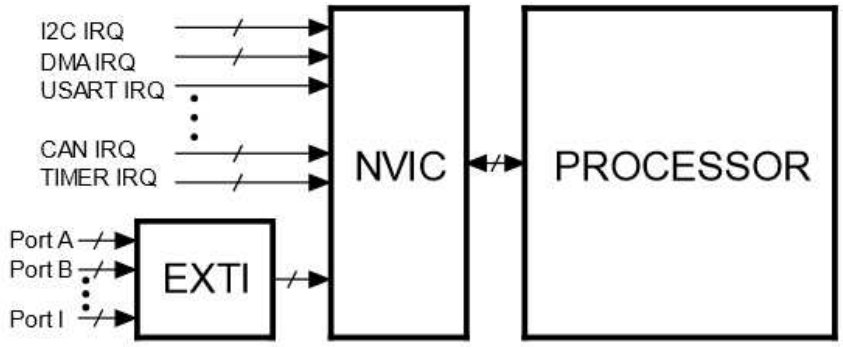
vhodni parameter dobi enumerator pina (številko pina), ki ga hočemo inicializirati. Vsi podatki o pinu se nato preberejo iz zgoraj omenjenih tabel opisov pinov. Kot prvo z ukazom `RCC_AHB1PeriphClockCmd` vklopimo uro na vodilu, nato v strukturi `GPIO_InitStructure` inicializiramo podatke o pinu. Funkcija in klic sta prikazana v Kodi 5.4.

```
void STM_EVAL_OutInit(OutSignal_TypeDef_t Out_signal)
{
  GPIO_InitTypeDef  GPIO_InitStructure;

  /* Enable AHB1 clock */
  RCC_AHB1PeriphClockCmd(OUT_CLK[Out_signal], ENABLE);

  /* Configure pin */
  GPIO_InitStructure.GPIO_Pin   = OUT_PIN[Out_signal];
  GPIO_InitStructure.GPIO_Mode  = GPIO_Mode_OUT;
  GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
  GPIO_InitStructure.GPIO_PuPd  = GPIO_PuPd_UP;
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
  GPIO_Init(OUT_PORT[Out_signal], &GPIO_InitStructure);
}
/** Function call */
STM_EVAL_OutInit(Y_DIRECTION);
```

Koda 5.4: Funkcija za inicializacijo izhodnih pinov.

5.1.2 Inicializacija vhodnih pinov

Mikrokrmilnik nam omogoča uporabo zunanjih prekinitvenih linij, zato smo le-te uporabili za priklop končnih stikal. Zunanje prekinitvene linije so razdeljene v dva sklopa. Prvi sklop zavzemajo zunanji pini (P0-P15 na vsakih vratih), drugi sklop zavzemajo dogodki, kot so prekinitev ure realnega časa, USB prekinitev, (I2C, DMA, USART prekinitve) ... Na Sliki 5.1 je prikazan poenostavljen diagram procesiranja prekinitvenih signalov.



Slika 5.1: Poenostavljen diagram procesiranja prekinitvenih signalov [28].

V krmilniku GPIO (angl. General Purpose Input/Output) imamo na voljo 16 prekinitvenih linij. Označene so od `EXTI_Line0` do `EXTI_Line15` in predstavljajo tudi številke pinov (PA0-PA15). To pomeni, da je PA0 pin povezan na EXTI_Line0 in PA15 pin povezan na EXTI_Line15. Prav tako so pini ostalih vrat povezani na iste zunanje prekinitvene linije, kar pomeni, da sočasno ne moremo uporabljati dveh istih pinov na eni prekinitveni liniji. Vsaka linija lahko sproži prekinitev na prvo, zadnjo ali obe fronti.

Inicializacija vhodnih pinov končnih stikal je prikazana v kodi 5.5. Od inicializacije izhodnih pinov se razlikuje v dodanih parametrih za omogočanje zunanje prekinitve. V spremenljivki `SWITCH_X_1_EXTI_LINE` definiramo uporabo zunanje prekinitvene linije 0, v `SWITCH_X_1_EXTI_PORT_SOURCE` definiramo vrata GPIOA, v `SWITCH_X_1_EXTI_PIN_SOURCE` definiramo pin 0 na prekinitveni liniji 0 in v `SWITCH_X_1_EXTI_IRQn` definiramo uporabo prekinitveno servisnega programa `EXTI0_IRQn`. Te definicije vhodnih pinov prav tako shranimo v tabele opisov pinov, ki jih potem uporabimo v funkciji za inicializacijo vhodnih pinov (Koda 5.6).

```
#define SWITCH_X_1_PIN              GPIO_Pin_0    /*!< Pin
    */
#define SWITCH_X_1_GPIO_PORT        GPIOA     /*!< Port */
#define SWITCH_X_1_GPIO_CLK         RCC_AHB1Periph_GPIOA
    /*!< Clock */
```

```
#define SWITCH_X_1_EXTI_LINE      EXTI_Line0        /*!<
   External interrupt line0 */
#define SWITCH_X_1_EXTI_PORT_SOURCE EXTI_PortSourceGPIOA
   /*!< External port source*/
#define SWITCH_X_1_EXTI_PIN_SOURCE EXTI_PinSource0    /*!<
   External pin source  */
#define SWITCH_X_1_EXTI_IRQn      EXTI0_IRQn    /*!< EXTI
   Line0 Interrupt */
```

Koda 5.5: Inicializacija vhodnih pinov končnih stikal.

Funkcija `STM_EVAL_SwitchInit` kot vhodni parameter dobi enumerator vhodnega pina (številko pina) in način, v katerega se mora pin inicializirati. Če je `Switch_Mode` enak SWITCH_MODE_GPIO se pin inicializira kot navaden vhodni pin, če pa je `Switch_Mode` enak SWITCH_MODE_EXTI, se pin inicializira kot zunanja prekinitvena linija. V prvem delu funkcije inicializiramo pin kot navaden vhodni pin, v drugem delu pa inicializiramo pin kot zunanjo prekinitveno linijo ter vklopimo definirane prekinitve.

```
void STM_EVAL_SwitchInit(Switches_TypeDef_t Switch,
   SwitchMode_TypeDef Switch_Mode)
{
  GPIO_InitTypeDef GPIO_InitStructure;
  EXTI_InitTypeDef EXTI_InitStructure;
  NVIC_InitTypeDef NVIC_InitStructure;

  /* Enable the BUTTON Clock */
  RCC_AHB1PeriphClockCmd(SWITCH_CLK[Switch], ENABLE);
  RCC_APB2PeriphClockCmd(RCC_APB2Periph_SYSCFG, ENABLE);

  /* Configure Switch pin as input */
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN;
  GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_DOWN;
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_2MHz;
  GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
```

```
  GPIO_InitStructure.GPIO_Pin = SWITCH_PIN[Switch];
  GPIO_Init(SWITCH_PORT[Switch], &GPIO_InitStructure);

  if (Switch_Mode == SWITCH_MODE_EXTI)
  {
    /* Connect Switch EXTI Line to Switch GPIO Pin */
    SYSCFG_EXTILineConfig(SWITCH_PORT_SOURCE[Switch],
        SWITCH_PIN_SOURCE[Switch]);

    /* Configure Switch EXTI line */
    EXTI_InitStructure.EXTI_Line =
        SWITCH_EXTI_LINE[Switch];
    EXTI_InitStructure.EXTI_Mode = EXTI_Mode_Interrupt;
    EXTI_InitStructure.EXTI_Trigger =
        EXTI_Trigger_Falling;//EXTI_Trigger_Rising_Falling;
    EXTI_InitStructure.EXTI_LineCmd = ENABLE;
    EXTI_Init(&EXTI_InitStructure);

    /* Enable and set Switch EXTI Interrupt */
    NVIC_InitStructure.NVIC_IRQChannel =
        SWITCH_IRQn[Switch];
    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority =
        0x05;
    NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0x05;
    NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;

    NVIC_Init(&NVIC_InitStructure);
  }
}
```

Koda 5.6: Funkcija za inicializacijo vhodnih pinov.

Ker zunanje linije uporabljajo prekinitve, moramo inicializirati tudi prekinitveni krmilnik NVIC. NVIC omogoča priklop do 240 prekinitvenih izvorov s 256 različnimi prioritetnimi nivoji. Procesorsko stanje je ob vstopu v prekinitveno servisni program avtomatsko shranjeno in ob izstopu avtomatsko

obnovljeno. Funkcija, ki bo obravnavala prekinitev, mora imeti enako ime, kot je zapisano v vektorski tabeli.

Tako v Kodi 5.6 definiramo strukturo `NVIC_InitStructure` ki jo inicializiramo z ustreznimi parametri. `NVIC_IRQChannel` nam določa, katero prekinitveno zahtevo bomo vklopili. Za pin 0 na vratih GPIOA je ta vrednost `EXTI0_IRQn`, za pin 1 `EXTI1_IRQn` ...

5.1.3 Inicializacija komunikacijskega vmesnika (USART)

Za namen povezovanja mikrokrmilnikov z drugimi napravami (računalniki, gsm moduli, bluetooth moduli ...) imajo mikrokrmilniki vgrajenih več vrst komunikacijskih vmesnikov. Eden od teh je USART [29] (angl. Universal synchronous/asynchronous receiver/transmitter), ki smo ga tudi uporabili za glavni komunikacijski kanal s krmilnim programom na računalniku. USART je sinhroni način komunikacije, pri kateri nimamo start in stop bitov, zato morata sprejemnik in oddajnik ostati v sinhronizaciji med celotnim sporočilom.

STM32F407 podpira štiri UART in štiri USART komunikacijske kanale. Za potrebe naše naprave smo izbrali USART6 kanal, ki se nahaja na pinih P6 in P7 na vratih GPIOC. Pri inicializaciji USART6 moramo kot prvo vklopiti uro vodila na vratih GPIOC, pinu PC6 in PC7 pa nastavimo alternativno funkcijo `GPIOC_AF_USART6`. Sledi inicializacija pinov, kjer jima nastavimo način delovanja na alternativnega z ukazom `GPIO_MODE_AF` (Koda 5.7).

```
void USART_GPIO_Configuration(void)
{
  GPIO_InitTypeDef GPIO_InitStructure;

  /* GPIOC clock enable */
  RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOC, ENABLE);

  /* Connect USART pins to AF */
  GPIO_PinAFConfig(GPIOC, GPIO_PinSource6,
     GPIO_AF_USART6);   // USART6_TX
```

```
  GPIO_PinAFConfig(GPIOC, GPIO_PinSource7,
     GPIO_AF_USART6);  // USART6_RX

  /* GPIO Configuration */
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_6 | GPIO_Pin_7;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;
  GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
  GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_2MHz;
  GPIO_Init(GPIOC, &GPIO_InitStructure);
}
```

Koda 5.7: Inicializacija USART6 pinov.

Nato inicializiramo USART6 parametre komunikacije in na koncu vklopimo USART6 prekinitev `USART_IT_RXNE`. Prekinitev se sproži, ko so podatki sprejeti in pripravljeni za branje. Prav tako nastavimo NVIC, saj za sprejem in oddajanje uporabljamo prekinitve. Prikaz inicializacije je v kodi 5.8.

```
void USART_Configuration(void)
{
  USART_InitTypeDef USART_InitStructure;

  /* UART6 clock enable */
  RCC_APB2PeriphClockCmd(RCC_APB2Periph_USART6, ENABLE);

  USART_InitStructure.USART_BaudRate = 115200;
  USART_InitStructure.USART_WordLength =
     USART_WordLength_8b;
  USART_InitStructure.USART_StopBits = USART_StopBits_1;
  USART_InitStructure.USART_Parity = USART_Parity_No;
  USART_InitStructure.USART_HardwareFlowControl =
     USART_HardwareFlowControl_None;

  USART_InitStructure.USART_Mode = USART_Mode_Rx |
     USART_Mode_Tx;
```

```
  USART_Init(USART6, &USART_InitStructure);

  /* Enables the USART6 receive interrupt */
  USART_ITConfig(USART6, USART_IT_RXNE, ENABLE);
  USART_Cmd(USART6, ENABLE);
}
void USART_NVIC_Configuration(void)
{
  NVIC_InitTypeDef NVIC_InitStructure;
  /* Configure USART6 interrupt */
  NVIC_InitStructure.NVIC_IRQChannel = USART6_IRQn;
  /* Set priority level */
  NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 3;
  NVIC_InitStructure.NVIC_IRQChannelSubPriority = 3;
  NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
  NVIC_Init(&NVIC_InitStructure);
  USART_Cmd(USART6, ENABLE);
}
```

Koda 5.8: Inicializacija USART6 serijske komunikacije.

5.1.4 Inicializacija časovnikov

Za potrebe krmiljenja treh osi v sistemu uporabljamo tri časovnike, s katerimi generiramo tri neodvisne PWM signale. STM32F407 ima dva naprednejša časovnika (TIM1 in TIM8), šest splošno namenskih časovnikov (TIM2, TIM3, TIM4, TIM5, TIM9, TIM14) in dva osnovna časovnika (TIM6, TIM7). Osnovna operacija časovnikov je spreminjanje vrednosti v registru števca, v našem primeru pa ga bomo uporabili za generiranje PWM signala. Uporabili smo TIM3 za X, TIM4 za Y in TIM1 za os Z. Vse tri časovnike smo inicializirali v PWM načinu, kjer je osnovna oziroma začetna frekvenca signala znašala 6kHz. Ta frekvenca se v času delovanja naprave spreminja, ker lahko na ta način zgladimo začetek in konec premika. Pred inicializacijo časovnika

moramo prav tako nastaviti pine, na katere bo PWM imel vpliv. Nastavimo jim alternativno funkcijo `GPIO_AF_TIMx` in postavimo jih v alternativni način delovanja, kar prikazuje Koda 5.9. Inicializacija časovnika in PWM-ja je prikazana v Kodi 5.10.

```
void TM_GPIO_Init(void)
{
  GPIO_InitTypeDef GPIO_InitStruct;

  /* Clock for GPIOD */
  RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOD, ENABLE);

  /* Alternating function for pin */
  GPIO_PinAFConfig(GPIOD, GPIO_PinSource15, GPIO_AF_TIM4);

  /* Set PWM pin for Y*/
  GPIO_InitStruct.GPIO_Pin = GPIO_Pin_15;
  GPIO_InitStruct.GPIO_OType = GPIO_OType_PP;
  GPIO_InitStruct.GPIO_PuPd = GPIO_PuPd_NOPULL;
  GPIO_InitStruct.GPIO_Mode = GPIO_Mode_AF;
  GPIO_InitStruct.GPIO_Speed = GPIO_Speed_100MHz;
  GPIO_Init(GPIOD, &GPIO_InitStruct);
}
```

Koda 5.9: Inicializacija pinov izhod PWM-jev.

```
void TM_TIMER_Init(void)
{
    TIM_TimeBaseInitTypeDef TIM_BaseStructTimY;

    /* Enable clock for TIM4 */
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM4, ENABLE);
  /* Set prescaler value */
    TIM_BaseStructTimY.TIM_Prescaler = 2;
    /* Count up */
```

```
    TIM_BaseStructTimY.TIM_CounterMode =
       TIM_CounterMode_Up;
    /* Set timer period */
    TIM_BaseStructTimY.TIM_Period = periodY; // default
       period set to 6000
    TIM_BaseStructTimY.TIM_ClockDivision = TIM_CKD_DIV1;
    TIM_BaseStructTimY.TIM_RepetitionCounter = 0;
    TIM_TimeBaseInit(TIM4, &TIM_BaseStructTimY);

    /* Set NVIC interrupt for TIM4 */
    NVIC_InitTypeDef NVIC_InitStructureY;

    NVIC_InitStructureY.NVIC_IRQChannel = TIM4_IRQn;
    NVIC_InitStructureY.NVIC_IRQChannelPreemptionPriority
       = 0;
    NVIC_InitStructureY.NVIC_IRQChannelSubPriority = 1;
    NVIC_InitStructureY.NVIC_IRQChannelCmd = ENABLE;
    NVIC_Init(&NVIC_InitStructureY);
    /* Set interrupt on timer update flag */
    TIM_ITConfig(TIM4, TIM_IT_Update, ENABLE);

    /* Set and enable PWM on TIM4 */
    TIM_OCInitTypeDef TIM_OCStruct;

    TIM_OCStructInit(&TIM_OCStruct);
    /* PWM mode 2 = Clear on compare match */
    TIM_OCStruct.TIM_OCMode = TIM_OCMode_PWM2;
    TIM_OCStruct.TIM_OutputState = TIM_OutputState_Enable;
    TIM_OCStruct.TIM_OCPolarity = TIM_OCPolarity_High;
    TIM_OCStruct.TIM_Pulse = PULSE_LENGTH; // pulse length
       is 100
    TIM_OC4Init(TIM4, &TIM_OCStruct);
    TIM_OC4PreloadConfig(TIM4, TIM_OCPreload_Enable);
}
```

Koda 5.10: Inicializacija časovnika TIM4 v PWM načinu za krmiljenje Y-osi.

5.2 Delovanje programa

Kot smo omenili v poglavju 5.1, je celotno delovanje programa realizirano z dvema opraviloma. Opravilo `cnc_SerCon_process` skrbi za komunikacijo s krmilnim programom na računalniku ter za posredovanje ukazov opravilu `cnc_CNC_process`, ki skrbi za pravilno krmiljenje naprave. Obe opravili delujeta po principu končnega avtomata in se vrtita v neskončni zanki. Imata svoj kontekst, v katerem imata shranjeno referenco na vrsto sporočil, svoje trenutno stanje in strukturo sporočila, ki ga lahko pošljeta oziroma sprejmeta. Struktura sporočila vsebuje dogodek, ki enolično opisuje sporočilo in je hkrati stanje avtomata opravila.

Sporočilne vrste so v sistemu FreeRTOS primarni način komunikacije med opravili. Vsakemu opravilu lahko določimo velikost pomnilnika, ki ga uporablja za sporočilno vrsto. Vrste omogočajo pošiljaje sporočila v načinu prvi noter in prvi ven FIFO (angl. First In First Out), pri čemer se sporočilo pošlje na konec vrste ter zadnji noter, prvi ven LIFO (angl. Last In First Out), pri čemer se sporočilo pošlje na začetek vrste. FreeRTOS vsebuje vse potrebne funkcije za upravljanje z vrstami. Najbolj pogoste so `xQueueCreate` (kreiranje vrste), `xQueueSendToBack` (pošiljanje sporočila na konec vrste), `xQueueReceive` (sprejetje sporočila iz vrste), `xQueueDelete` (brisanje vrste). Vsebuje tudi funkcije, s katerimi lahko varno pošljemo sporočilo iz prekinitveno servisne rutine.

Vsi ukazi, ki pridejo iz krmilnega programa, se posredujejo v sporočilno vrsto opravila `cnc_SerCon_process`. To opravilo sporočilo pregleda in izvede ustrezne akcije. V Kodi 5.11 sta prikazana prejemanje ukazov in posredovanje le-teh opravilu `cnc_SerCon_process`.

```
/* max string length of received command */
#define MAX_STRLEN 64
/* received command holder */
char received_string[MAX_STRLEN];
/* string length counter */
static uint8_t cnt = 0;
```

```
char t = '\0';
void USART6_IRQHandler(void)
{
  if( USART_GetITStatus(USART6, USART_IT_RXNE) )
  {
    /* Turn on RX led */
    STM_EVAL_LEDOn(LED_ORANGE);
    /* character from USART6 data register */
    t = USART6->DR;

    if( (t != '\n') && (cnt < MAX_STRLEN) )
    {
      received_string[cnt] = t;
      cnt++;
    }
    /* terminating command character */
    if (t == '\r')
    {
      /* initialize cnc_SerCon_msg_t struct */
      cnc_SerCon_msg_t msg;

      /* set event to serial connection received string */
      msg.event = SERCON_RX_STRING;
      memset(&msg.data.data, '\0', MAX_STRLEN + 1);
      strncpy((char*)&msg.data.data,
          (char*)&received_string,
          strlen(received_string));
      /* send message to cnc_SerCon_process task */
      xQueueSend(pSerConCtx->xQueue, &msg, 5);

      /* reset counter an memset received string to zero */
      cnt = 0;
      memset(&received_string, '\0', MAX_STRLEN + 1);
    }
    /* clear interrupt flag */
    USART_ClearITPendingBit(USART6, USART_IT_RXNE);
  }
```

```
}
```

Koda 5.11: USART6 prekinitveno servisna procedura.

V Kodi 5.12 je prikazano sprejetje sporočila iz njegove vrste. To sporočilo se potem glede na dogodek pregleda in izvede nadaljnje ukaze. Na primer če iz krmilnega programa pošljemo sporočilo "null\r\n", se to sporočilo prebere in posreduje opravilu `cnc_CNC_process` s kodo dogodka `CNC_NULL`. Opravilo `cnc_CNC_process` to sporočilo sprejme, ponastavi števce ter parametre na privzete vrednosti, nastavi pravilne smeri premikov ter zažene časovnike za generiranje PWM signalov.

```
...
while( pCtx->status.alive )
{
    rc = xQueueReceive(pCtx->xQueue, &msg, 10);
    if( rc == CNC_OK )
  {
    switch( msg.event )
    {
      case SERCON_RX_STRING:
      {
        /* Turn off RX led */
        STM_EVAL_LEDOff(LED_ORANGE);
        if (strncmp(msg.data.data, "null", 4) == 0)
        {
          cnc_CNC_msg_t msg;

          printf("Nulling in progress...\r\n");
          /* set event code and send message to cnc task */
          msg.event = CNC_NULL;
          xQueueSend(pCNCCtx->xQueue, &msg, 5);
        }
...
```

Koda 5.12: cnc_SerCon_process sprejetje sporočila in izvedba ukaza.

Ko `cnc_CNC_process` opravilo dobi sporočilo s kodo dogodka CNC_PROCESS_NULL, pošlje sporočilo cnc_SerCon_process opravilu s kodo dogodka CNC_READY, ki krmilnemu programu sporoči, da je pozicioniranje na privzeto pozicijo zaključeno. Za sporočanje v smeri mikrokrmilnik-računalnik, smo priredili funkcijo `printf` tako, da znake zapisuje v krožno tabelo predefinirane velikosti. Ima svoj pisalni kazalec, ki ga premika ob vsakem zapisu. Če pride kazalec do dolžine tabele, ga postavimo na začetek le-te. Vsakič, ko se celoten ukaz zapiše do konca, vklopimo `USART_IT_TXE` prekinitev za prenos. V USART6 prekinitveno servisni proceduri beremo zastavico USART_FLAG_TXE, dokler ne preberemo oziroma pošljemo celotnega

ukaza, nato izklopimo prekinitev in prenos je zaključen. Postopek zapisa v krožno tabelo in pošiljanje ukaza sta prikazana v Kodi 5.13;.

```
...
  for(i = 0; i < len; i++)
    {
      out_string[w_pos++] = *ptr++;
      if (w_pos == OUT_STRING_LEN) {
        w_pos = 0;
      }
    }
    USART_ITConfig(USART6, USART_IT_TXE, ENABLE);
...

...
  if (USART_GetITStatus(USART6, USART_IT_TXE))
  {
    if (out_string[r_pos] == '\0')
    {
      USART_ITConfig(USART6, USART_IT_TXE, DISABLE);
    }
    else
    {
      USART_SendData(USART6, out_string[r_pos]);
      out_string[r_pos++] = '\0';
      if (r_pos == OUT_STRING_LEN)
      {
        r_pos = 0;
      }
    }
  }
...
```

Koda 5.13: Zapis znakov v krožno tabelo in pošiljanje sporočila.

Koordinate vrtanja krmilni program sporoča z ukazom "`COR: X<Xkoor>Y<`

`Ykoor>`". Pri tem sta Xkoor in Ykoor šestmestni števili, ki opisujeta koordinato na 1/10000 milimetra natančno. Ker pa za vrtanje ne potrebujemo tako velike resolucije, koordinate delimo s faktorjem 1000 in tako dobimo resolucijo 1/10 milimetra. Ta ukaz opravilo cnc_SerCon_process razčleni in iz njega izlušči le številske parametre.

Ker pri krmiljenju naprave uporabljamo mikrokorakanje, za en polni obrat motorja potrebujemo 3200 korakov. Navojne palice imajo korak 1,75 mm, kar pomeni, da za en milimeter potrebujemo 1829 korakov (5.1).

$$st\_korakov\_za\_1mm = \frac{st\_korakov\_za\_en\_obrat\_motorja}{korak\_navojne\_palice} \tag{5.1}$$
$$1829 = \frac{3200}{1,75}$$

Koda 5.14 prikazuje razčlenitev koordinat iz ukaza in pošiljanje le-teh cnc_CNC_process opravilu. Opravilo cnc_CNC_process to sporočilo sprejme, nastavi ustrezne parametre in vklopi ustrezne časovnike za generiranje signala za krmiljenje ustrezne osi.

```
#define OFFSET_COUNT 1829
...
  char X[6], Y[6];
  UINT32 i = 0, x, y;
  cnc_CNC_msg_t msg1;

  for (i = 6; i < 12; i++)
  {
    X[i-6] = msg.data.data[i];
    Y[i-6] = msg.data.data[i+7];
  }

  x = (INT32)(atoi((char*)&X[0]) * OFFSET_COUNT) / 1000;
  y = (INT32)(atoi((char*)&Y[0]) * OFFSET_COUNT) / 1000;

  msg1.event = CNC_PROCESS_START;

  msg1.data.arr.status = 2;
```

```
  msg1.data.arr.x = x;
  msg1.data.arr.y = y;

  xQueueSend(pCNCCtx->xQueue, &msg1, 10);
...
```

Koda 5.14: Razčlenitev ukaza koordinat.

V Kodi 5.15 je prikazano delovanje prekinitveno servisne procedure časovnika TIM3. V proceduri od globalne spremenljivke `counterX` odštevamo oziroma prištevamo vrednost ena, kar nam ponazarja korakanje motorja. Operacija je odvisna od smeri, v katero se mora obdelovalno orodje premakniti. Ko je vrednost counterX enaka endX, v kateri je shranjena x koordinata trenutne luknje, časovnik ustavimo ter pošljemo sporočilo opravilu cnc_CNC_process, da je premik v smeri x končan. Ko opravilo dobi potrditev od časovnika za premik X in Y-osi, da premik ustreza zahtevanim koordinatam, se vklopita obdelovalno orodje ter časovnik za premik Z-osi. Ko opravilo dobi potrditev, da je koordinata obdelana, pošlje krmilnemu programu potrditev in zahtevo za novo koordinato.

```
void TIM3_IRQHandler()
{
  if (TIM_GetITStatus(TIM3, TIM_IT_Update) != RESET)
  {
    ...
    else
    {
      if (xDetect)
      {
        if (counterX == endX)
        {
          cnc_CNC_msg_t     msg;

          TM_PWM_StatX(DISABLE);
          periodX = DEFAULT_PERIOD;
```

```
            xDetect = FALSE;

            msg.event = CNC_PROCESS_CHECK;
            msg.data.intMsg.who = TIMER_X;
            msg.data.intMsg.what = TIMER_X_FINISH;
            xQueueSend(pCNCCtx->xQueue, &msg, 10);

            TIM_ClearITPendingBit(TIM3, TIM_IT_Update);
            return;
          }

          if (STM_EVAL_OutGetState(X_DIRECTION1) == RESET)
          {
            counterX--;
          }
          else
          {
            counterX++;
          }
        }
      }
      TIM_ClearITPendingBit(TIM3, TIM_IT_Update);
    }
}
```

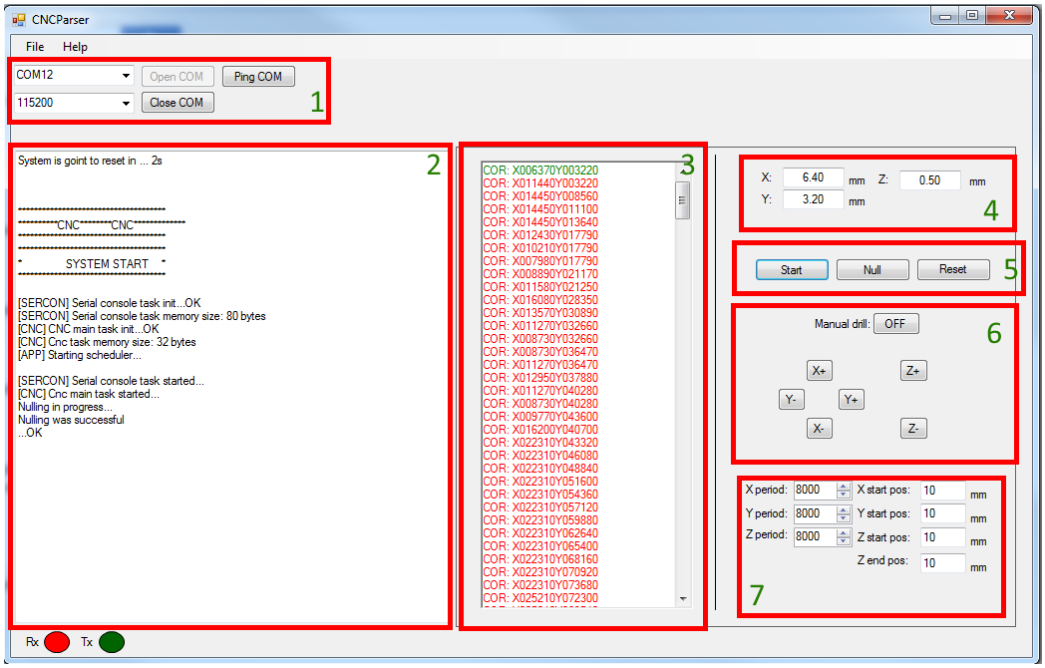
Koda 5.15: Prekinitveno servisna procedura časovnika TIM3.

5.3 Program za krmiljenje naprave

Da bi lahko upravljali napravo ter spremljali njeno trenutno stanje, potrebujemo krmilni oziroma nadzorni program. Omogočati nam mora nadzor nad operacijami naprave in vpogled v trenutno stanje. Napisan je v programskem jeziku C# z uporabo ogrodja .Net Framework 4. Zasnovan je tako, da je z njim možno izvesti vse možne operacije, ki jih naprava trenutno podpira. Na

sliki 5.2 je prikazan uporabniški vmesnik programa.

V okvirju 1 imamo možnost izbire komunikacijskih vrat ter hitrosti, z gumbom "Open COM" pa odpremo serijsko komunikacijo. Okvir 2 se uporablja za prikaz sporočil iz mikrokrmilnika, v okvirju 3 imamo z zeleno barvo prikazane že obdelane koordinate, z rdečo pa neobdelane. Ko se koordinata X obdela, se avtomatsko obarva v zeleno barvo. V okvirju 4 imamo trenutne



Slika 5.2: Krmilni program.

pozicije obdelovalnega orodja vseh treh osi. V okvirju 5 imamo gumbe za ponovni zagon naprave, za nuliranje naprave ter za zagon avtomatskega načina delovanja. V okvirju 6 imamo gumbe za ročni način premika obdelovalnega orodja ter možnost zagona vrtalnika. V okvirju 7 imamo možnost ročne nastavitve hitrosti premikanja posamezne osi ter možnost ročne nastavitve nulte pozicije obdelovalnega orodja.

Ko uporabnik zažene program, najprej izbere serijska vrata ter le-ta odpre. Nadaljuje z gumbom "Null", kar sproži pozicioniranje naprave v začetno pozicijo. Obdelovalno orodje se iz trenutne pozicije premakne v privzeto po-

zicijo, ki je definirana na mikrokrmilniku. Nato odpre datoteko s koordinatami tiskanega vezja, ki se v program naložijo v tabelo ter v okno številka 3. Ko pritisne gumb "Start", se prične avtomatski način delovanja. Na mikrokrmilnik se pošlje prva koordinata iz tabele. Koordinate se na napravo pošiljajo z ukazom `COR: X<koordinata>Y<koordinata>` pri čemer je koordinata šestmestno število. Ko je koordinata na napravi obdelana, naprava sporoči programu, naj pošlje naslednjo. Ko program dobi potrditev o končani obdelavi, vrstico obdelane koordinate obarva zeleno. Ta postopek traja, dokler se ne obdelajo vse koordinate. Ko program nima več koordinat za obdelavo, pošlje ukaz "end". Po tem ukazu se obdelovalno orodje premakne v začetno pozicijo, s čimer je avtomatski postopek končan.

Poglavje 6

Celotna naprava

Za sestavo naprave je bilo potrebnega veliko natančnega dela, posebno pri sestavi in načrtovanju mehanskega dela. Mehanski del sestavlja več komponent, ki smo jih natančno povezali v celotno napravo. Pri postavitvi končnih stikal smo morali biti pozorni na maksimalne možne premike obdelovalnega orodja po posamezni osi. V nasprotnem primeru končna stikala ne bi koristila svojemu namenu, saj se naprava v primeru napake ne bi ustavila. Obdelovalno orodje premikajo štirje koračni motorji. Zaradi manjše obremenitve premika imata Y in Z-os po en motor, X-os pa dva. X-os je nosilna os preostalega dela naprave, zaradi tega smo za premik uporabili dva motorja. Razvojna plošča SMT32F4Discovery je na krmilno vezje pritrjena prek svojih vhodno/izhodnih pinov.

Celotno napravo krmili operacijski sistem za delo v realnem času FreeRTOS, nameščen na mikrokrmilnik STM32F407. Implementirani ima dve opravili, ki omogočata celovit nadzor nad napravo. Opravili med seboj komunicirata prek sporočilne vrste. Prvo opravilo nadzira vhodno/izhodne pine ter časovnike, ki generirajo PWM signale za krmiljenje koračnih motorjev. Drugo opravilo skrbi za komunikacijo s krmilnim programom ter posredovanje sporočil iz krmilnega programa prvemu opravilu.

Program za krmiljenje omogoča uporabniku nadzor nad napravo, ročno ali avtomatsko krmiljenje ter pregled trenutnega stanja naprave. Omogoča

uvoz koordinat tiskanega vezja iz programa Sprint-Layout, ki se ob uvozu pretvorijo v prilagojen format za našo napravo. Ko uporabnik sproži avtomatski način delovanja, se koordinate začnejo prenašati na mikrokrmilnik. Obdelovalno orodje prične vrtanje, ko poziciji na osi X in Y ustrezata sprejetim koordinatam. Obdelava koordinate je končana, ko je premik po Z-osi enak začetni poziciji. Nato se pošljeta povratna informacija o uspešni obdelavi ter zahteva za novo koordinato. Ob končani obdelavi koordinat se obdelovalno orodje premakne v začetno pozicijo.

Poglavje 7

Sklepne ugotovitve

V diplomskem delu je bila v celoti realizirana ideja o izdelavi naprave CNC do končnega delujočega prototipa. V delu smo predstavili strojni del naprave, mehanske elemente ter uporabljene materiale ter program na mikrokrmilniku in sam program za krmiljenje. Glavni in najpomembnejši del naprave je razvojna plošča STM32F4Discovery, na kateri je mikrokrmilnik STM32F407. Na mikrokrmilnik smo implementirali operacijski sistem za delo v realnem času FreeRTOS, v katerem smo z uporabo opravil dosegli želen način delovanja. Za krmiljenje naprave smo v programskem jeziku C# razvili program, ki uporabniku omogoča nadzor in krmiljenje naprave.

Pri razvoju pa ni šlo brez težav. Pri prvem prototipu smo uporabili koračne motorje iz starih tiskalnikov. Problem se je pojavil, ker so bili vsi koračni motorji drugačni. Vsak je imel drugačen korak premika, ter različno število priklopnih žic. Prav tako smo pri prvem prototipu za krmiljenje koračnih motorjev uporabili krmilnik za koračne motorje z uporabo čipov L297-L298. Zaradi težav, ki smo jih imeli s krmiljenjem štirih popolnoma različnih koračnih motorjev, smo pri drugem prototipu uporabili komplet štirih koračnih motorjev NEMA17 s krmilniki DQ420MA. Uporaba le-teh nam je omogočila uporabo mikrokorakanja in posledično večje natančnosti pri premikih obdelovalnega orodja.

Kot vsak prototip naprave ima tudi naša naprava pomanjkljivosti. Ena večjih napak je napačna zasnova ogrodja naprave, saj le-tega ne moremo uporabiti za npr. rezkalno napravo CNC ali napravo za 3D-tisk. Napaka je pri zasnovi Z-osi, saj je maksimalna možna globina premika po Z-osi 5 cm. Problem naprave je tudi v netočnih premikih po X-osi, saj premike opravljata dva koračna motorja, med katerima lahko pride do nesinhronizacije. Pojavila se je tudi težava pri vrtanju, saj je Z-os slabo pritrjena na samo ogrodje in se pri vrtanju premika.

Napravo bi lahko še izboljšali. Lahko bi strojni del in ogrodje nadgradili, da bi lahko napravo uporabili še za druge namene. Izboljšave so možne tudi na področju premika posameznih osi. Lahko bi uporabili navojne palice s trapeznim navojem, ki so veliko bolj kompaktne in omogočajo boljšo natančnost premika. Programski del bi lahko nadgradili na ta način, da bi podpiral G-kodo [30], ki je najbolj razširjena med napravami CNC.

Literatura

[1] STM32F4Discovery. [Online]. Dosegljivo:
http://www.st.com/web/catalog/tools/FM116/SC959/SS1532/PF252419
[Dostopno 14. 5. 2015].

[2] CNC. [Online]. Dosegljivo:
http://en.wikipedia.org/wiki/CNC_router [Dostopno 31. 3. 2015].

[3] CAD/CAM. [Online]. Dosegljivo:
http://en.wikipedia.org/wiki/Computer-aided_design
[Dostopno 6. 5. 2015].

[4] NC. [Online]. Dosegljivo:
http://en.wikipedia.org/wiki/Numerical_control [Dostopno 6. 5. 2015].

[5] Slika točkovnega krmiljenja. [Online]. Dosegljivo:
http://www2.sts.si/arhiv/cncpro/images/diag1.gif
[Dostopno 14. 5. 2015].

[6] Slika linijskega krmiljenja. [Online]. Dosegljivo:
http://www2.sts.si/arhiv/cncpro/images/diag2.gif
[Dostopno 14. 5. 2015].

[7] Slika konturnega krmiljenja. [Online]. Dosegljivo:
http://www2.sts.si/arhiv/cncpro/images/diag4.gif
[Dostopno 14. 5. 2015].

[8] Programski jezik C. [Online]. Dosegljivo:
http://en.wikipedia.org/wiki/C_%28programming_language%29
[Dostopno 14. 5. 2014].

[9] PDP-11. [Online]. Dosegljivo:
http://en.wikipedia.org/wiki/PDP-11 [Dostopno 31. 3. 2015].

[10] Programski jezik C#. [Online]. Dosegljivo:
http://en.wikipedia.org/wiki/C_Sharp_%28programming_language%29
[Dostopno 14. 5. 2015].

[11] FreeRTOS. [Online]. Dosegljivo:
http://en.wikipedia.org/wiki/FreeRTOS [Dostopno 31. 3. 2015].

[12] GPOS. [Online]. Dosegljivo:
http://encyclopedia2.thefreedictionary.com/General-
purpose+operating+system [Dostopno 31. 3. 2015].

[13] Sprint-Layout. [Online]. Dosegljivo:
http://www.abacom-online.de/uk/html/sprint-layout.html
[Dostopno 31. 3. 2015].

[14] CooCox IDE. [Online]. Dosegljivo:
http://www.coocox.org [Dostopno 31. 3. 2015].

[15] Microsoft Visual Studio 2010. [Online]. Dosegljivo:
http://en.wikipedia.org/wiki/Microsoft_Visual_Studio#Visual_Studio_2010
[Dostopno 31. 3. 2015].

[16] PCB. [Online]. Dosegljivo:
http://en.wikipedia.org/wiki/Printed_circuit_board
[Dostopno 6. 5. 2015].

[17] Excellon format. [Online]. Dosegljivo:
http://en.wikipedia.org/wiki/Excellon_format.
[Dostopano 31. 3. 2015].

[18] Excellon Automation Co. (2005) Excellon ukazi. [Online]. Dosegljivo: http://www.excellon.com/manuals/program.htm. [Dostopano 31. 3. 2015].

[19] NEMA. [Online]. Dosegljivo: http://reprap.org/wiki/NEMA_17_Stepper_motor [Dostopno 6. 5. 2015].

[20] Zgradba koračnega motorja in priklop signalnih linij. [Online]. Dosegljivo:
http://goo.gl/HV5k5i
[Dostopno 19. 5. 2015].

[21] Gonilnik DQ420MA. [Online]. Dosegljivo:
http://www.wantmotor.com/ProductsView.asp?id=273&pid=82
[Dostopno 14. 5. 2015].

[22] Napetostni regulator LM805. [Online]. Dosegljivo:
https://goo.gl/f5dklT [Dostopno 18. 5. 2015].

[23] Optični izolator. [Online]. Dosegljivo:
http://en.wikipedia.org/wiki/Opto-isolator [Dostopno 14. 5. 2015].

[24] STM32. [Online]. Dosegljivo:
http://en.wikipedia.org/wiki/STM32 [Dostopno 31. 3. 2015].

[25] J. Yiu, *"The Definitive Guide to ARM Cortex-M3 and Cortex-M4 Processors"*, ARM Ltd., Cambridge, UK, 2014

[26] ST-LINK/V2. [Online]. Dosegljivo:
http://www.st.com/web/catalog/tools/PF219866
[Dostopno 14. 5. 2015].

[27] PWM. [Online]. Dosegljivo:
http://en.wikipedia.org/wiki/Pulse-width_modulation
[Dostopno 30. 4. 2015].

[28] Poenostavljen diagram procesiranja prekinitvenih signalov. [Online]. Dosegljivo:
https://bluetechs.wordpress.com/zothers/x/interrupts-2/
[Dostopno 18. 5. 15].

[29] USART. [Online]. Dosegljivo:
http://en.wikipedia.org/wiki/Universal_asynchronous_receiver/transmitter
[Dostopno 4. 5. 2015].

[30] G-koda. [Online]. Dosegljivo:
http://en.wikipedia.org/wiki/G-code [Dostopno 6. 5. 2015]